EXHIBIT 6

UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MIDLAND/ODESSA DIVISION

VIRTAMOVE, CORP., Plaintiff, v. AMAZON.COM, INC.; AMAZON.COM SERVICES LLC; AND AMAZON WEB SERVICES, INC., Defendants.	Case No. 7:24-CV-00030-ADA-DTG

PLAINTIFF'S RESPONSIVE
CLAIM CONSTRUCTION BRIEF

TABLE OF CONTENTS

TABLE OF AUTHORITIES

Cases

Statutes

Amazon contends that all asserted claims of both patents asserted in this action are invalid for indefiniteness. Amazon does not and cannot prove invalidity by clear and convincing evidence as required by the law, and its implicit motion for summary judgment should be denied. Amazon also proposes several constructions impermissibly changing the patent scope from the plain and ordinary meaning set out by the applicant and issued by the U.S. Patent and Trademark Office. These constructions should be rejected for the reasons explained below.

I. Amazon has not shown invalidity of the '814 Patent claims by clear and convincing evidence

A. "disparate computing environments" (claim 1)

Plaintiff's Proposed Construction	Defendants' Proposed Construction
Environments run by standalone or unrelated computers	Indefinite

In the co-pending action *VirtaMove, Corp. v. Google LLC*, Case No. 7:24-cv-00033-ADA-DTG (W.D. Tex.), VirtaMove has proposed construing "disparate computing environments" as "environments run by standalone or unrelated computers." For consistency across the two actions, VirtaMove proposes the same construction here. This proposal is closely based on the same passage from the specification that Amazon relies on as "lexicography" and that forms the basis of Amazon's indefiniteness challenge, which the parties also discussed in their pre-filing meet and confer process. *See* Dkt. 71 at 3 (alleging indefiniteness of: "Environments where computers are stand-alone or where there are plural computers and where they are unrelated.").

Amazon presents two different arguments in favor of indefiniteness. First, that the "unrelatedness" of two computers is indefinite, and second, that the claim ***requires*** the computers must be "related," creating a contradiction. As discussed below, these arguments fail because they ignore the fact that a POSITA would understand the scope of "standalone" computers.

To Amazon's first point, Amazon contends that the claim is indefinite, alleging that "'unrelated' is subjective; people may reasonably disagree about whether two things are related or unrelated." Dkt. 71 at 4. But this entirely ignores the remainder of the claim, which Amazon ***admits*** does not allow for two computers to be "unrelated" because they must be "part of a single 'system.'" In other words, Amazon acknowledges that the "unrelated" portion of the specification's description of "disparate computing environments" cannot fit into the broader context of the claim language. Because the claim ***undisputedly*** cannot extend to "unrelated" computers, only the "standalone" portion of that description could be relevant to the scope of the claims as a whole. *See Phillips v. AWH Corp.*, 415 F.3d 1303, 1313 (Fed. Cir. 2005) ("Importantly, the person of ordinary skill in the art is deemed to read the claim term … in the context of the particular claim in which the disputed term appears….").

Accordingly, the only relevant inquiry (in the context of the claim as a whole) is whether environments run by ***standalone*** computers is indefinite. Amazon presents no evidence that a POSITA would be unable to understand the boundaries of standalone computers, which is a common phrase used to indicate the ability of computers to operate independently of each other. Amazon's narrow focus on computers being "unrelated" (a scenario that Amazon acknowledges is simply inapplicable in the context of the asserted claims) ignores whether "standalone" computers can be understood to a POSITA, and Amazon presents no evidence at all that standalone computers would not be understood.

Amazon also alleges in passing that "[n]othing in the patent or prosecution history explains how multiple computers can be part of the same system and yet be stand-alone or unrelated." Dkt. 71 at 4. As an initial matter, it is ***Amazon's*** burden to show indefiniteness by clear and convincing evidence; it is not VirtaMove's burden to prove definiteness as Amazon implies. And in any event,

two computers can be part of the claimed "system" and still be capable of operating independently, and Amazon presents no evidence to the contrary.

B. "container" (claim 1)

Plaintiff's Proposed Construction	Defendants' Proposed Construction
No construction necessary; plain and ordinary meaning. In the alternative: An aggregate of files required to successfully execute a set of software applications on a computing platform. Each container for use on a server is mutually exclusive of the other containers, such that read/write files within a container cannot be shared with other containers.	An aggregate of files required to successfully execute a set of software applications on a computing platform is referred to as a container. Each container for use on a server is mutually exclusive of the other containers, such that read/write files within a container cannot be shared with other containers. A container comprises one or more application programs including one or more processes, and associated system files for use in executing the one or more processes; but containers do not comprise a kernel; each container has its own execution file associated therewith for starting one or more applications.

The '814 Patent specification includes a broad explanation of how a "container" fits within the context of the claimed invention. Although described as a "definition," in substance the patentee provided an encyclopedia entry, which cannot reasonably be interpreted as pure lexicography, and which would serve only to confuse the jury by substituting a single word in a claim with nearly 100 words of redundant examples of how containers may be implemented. Indeed, Amazon itself omits entire sentences from the supposed "definition" set forth in the specification, confirming that a POSITA would not understand the entirety of its discussion of "container" to be lexicography.

Nor could Amazon have shown that the "exacting" standard for lexicography is met. "To act as its own lexicographer, a patentee must 'clearly set forth a definition of the disputed claim term' other than its plain and ordinary meaning." *Thorner v. Sony Computer Ent. Am. LLC*, 669 F.3d 1362, 1365 (Fed. Cir. 2012). And "[t]he standard for disavowal of claim scope is similarly

exacting. *Id.* The fact that high bar for lexicography is not met here is confirmed not only by the non-definitional nature of the specification's discussion of a "container," but also by the fact that two different defendants attempt to apply the alleged "lexicography" in completely different ways.

In particular, Google LLC applies the alleged "lexicography" of the specification to provide a ***substantially different*** proposed "definition" of "container." *VirtaMove Corp. vs. Google LLC*, Case No. 7:24-cv-00033-DC-DTG, Dkt. No. 63 at 8-10 (W.D. Tex. Oct. 22, 2024). The ***only*** overlap between Amazon's and Google's proposed constructions is the first sentence "An aggregate of files…" and the sentence "Each container for use on a server is mutually exclusive…." *Id.* Google does not agree with Amazon that the "execution file" language must be included, except as a fallback position. These disagreements confirm that the entire specification's explanation of containers need not be part of the construction of "container"

Regardless of how the Court construes "container," Plaintiff requests that the Court enter identical constructions in both the Google and Amazon actions. Plaintiff believes that the plain and ordinary meaning of "container" applies and is generally consistent with the only two sentences that both Google and Amazon have ***both*** proposed as being definitional: "An aggregate of files…" and "Each container for use on a server is mutually exclusive…." However, as explained below, only "An aggregate of files" would be potentially helpful in a construction for a jury, because the "mutually exclusive…" discussion is redundant with other claim language. This limitation, as well as other limitations Amazon seeks to read into the claim, are addressed below.

i. Redundant language need not and should not be included.

It is black letter law that claim construction "is not an obligatory exercise in redundancy." *U.S. Surgical Corp. v. Ethicon, Inc.*, 103 F.3d 1554, 1568 (Fed. Cir. 1997). Most of the proposed definition here is redundant of existing claim language. Including two slightly different phrasings

of the same concept would at best confuse the jury, and these redundant sentences should not be included.

"*Each container for use on a server is mutually exclusive of the other containers, such that read/write files within a container cannot be shared with other containers.*" This sentence is redundant of other express claim language. Claim 1 already recites "a plurality of secure containers of application software" wherein "the application software ***cannot be shared between the plurality of secure containers*** of application software." Thus, the claim already makes clear which portion of the container is mutually exclusive (i.e., cannot be shared with) other containers, and the "mutually exclusive" portion of Amazon's construction would inject confusion and surplusage.

"*A container comprises one or more application programs including one or more processes, and associated system files for use in executing the one or more processes*…" This phrase is redundant with the remainder of Claim 1, which already recites "each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications." Of course application programs include a "process"—otherwise the program would be unable to ***do*** (or "process") anything. Amazon presents no plausible reason why this language would add anything meaningful beyond the express claim language itself.

"*[B]ut containers do not comprise a kernel…*" Claim 1 already recites "the containers of application software excluding a kernel." As with this and the other redundant terms, the only possible purpose for a construction of "container" to include this requirement would be to cause redundancy and confuse the jury.

ii. "[e]ach container ha[ving] its own execution file" is expressly claimed in dependent claim 2, and should not be imported into independent claim 1.

"*[E]ach container has its own execution file associated therewith for starting one or more applications.*"

In one non-limiting embodiment, the '814 Patent specification describes a particular type of configuration data, within a container, that "defines how applications… are started… includ[ing] the definition of a first program to start when a container is installed on a compute platform 40," as shown in Figure 7. '814 Patent at 9:8-19 and Fig. 7. As the patent explains, this embodiment "***allows for any number of applications to be started with a container association.***" *Id.* at 9:18-19.

Claim 2 expressly recites this aspect of the preferred embodiment in specifying that "each container has an execution file associated therewith for starting the one or more applications."

Under Amazon's proposal, the scope of claims 1 and 2 would be literally coextensive, because the proposed construction requiring a container to have an "execution file associated therewith for starting one or more applications" is identical to the language of claim 2. Amazon also provides no justification for a construction which would, again, be redundant with the express language of claim 2.

II. Amazon has not shown invalidity of the '058 Patent claims by clear and convincing evidence

A. "critical system elements" (claim 1)

Plaintiff's Proposed Construction	Defendants' Proposed Construction
Any service or part of a service, "normally" supplied by an operating system, that is critical to the operation of a software application.	Indefinite

Contrasting with Amazon's other "lexicography" proposals, the '058 Patent does provide an unambiguous definition of the phrase "critical system element[s]," stating what a CSE is rather than providing examples or embodiments. The definition has two basic requirements: first, the CSE is "'normally' supplied by an operating system"; second, it is "critical to the operation of a software application."

Regarding the former requirement, the patent specification provides further context, explaining: "It is traditionally the task of an operating system to provide mechanisms to safely and effectively control access to shared resources. In some instances the centralized control of elements, critical to software applications, hereafter called critical system elements (CSEs)[,] creates a limitation caused by conflicts for shared resources." '058 Patent at 1:22-27. This illustrates the conventional arrangement wherein CSEs are "normally" provided by an operating system (i.e., they are provided by the operating system if the structure of the operating system is not modified beyond its default operation). The specification also provides contrasting examples of the "invention," consistent with the claims, where "some system elements that are critical to the operation of a software application *are replicated from kernel mode, into user mode*…. These system elements are contained in a shared library." *Id.* at 9:15-19 (emphasis added). The specification parallels the claim requirements and confirms that the OSCSEs recited in limitation 1(b) generally correspond to the operation of a conventional system (where the operating system provides the critical system elements), whereas the SLCSEs of limitation 1(c) generally correspond to a non-conventional aspect of the claimed invention (where the critical system elements are stored in a shared library, outside of the operating system).

Amazon's indefiniteness challenge stems from the assumption that a POSITA would be unable to determine "whether or not a service was normally found in operating systems." Dkt. 71

at 8. But Amazon gives no explanation why a survey of other operating systems could plausibly be relevant here. Rather, the relevant determination focuses on the only operating system relevant to the claim scope: the claimed operating system of limitation 1(b).

Second, Amazon's attack on the word "critical" fails. Amazon poses a series of hypothetical questions divorced from the intrinsic record. Dkt. 71 at 8. References to "crashing" or "perform[ing] unreliably or incorrectly," with no source other than attorney or expert imagination, do not illuminate the potential claim scope. Neither is Amazon assisted by citation to a dictionary definition that Amazon concedes does not actually define "critical" (but instead defines "criticality," which is a term that encompasses a "range" of "criticality") and that Amazon does not even attempt to link to the claim or specification context.

Rather, the Court should look to the text of the patent specification itself, the intrinsic evidence that is the best guide to the patent's meaning. *See, e.g.*, *Phillips v. AWH Corp.*, 415 F.3d 1303, 1319 (Fed. Cir. 2005) (extrinsic evidence, such as expert reports, "is unlikely to result in a reliable interpretation of patent claim scope unless considered in the context of the intrinsic evidence."); *OSRAM GmbH v. Int'l Trade Comm'n*, 505 F.3d 1351, 1356 (Fed.Cir.2007) ("The patent specification is the primary resource for determining how an invention would be understood by persons experienced in the field.").

The '058 Patent provides numerous examples of critical system elements, more than sufficient to illustrate what elements are "critical." First, the specification discusses "a TCP/IP stack," which a POSITA would readily recognize as the core network protocols used for Internet communication. '058 Patent at 5:41-53. The TCP/IP stack is plainly critical to any application that uses Internet communication. The next examples are additional network services, "including TCP/IP, Bluetooth, ATM; or message passing protocols." *Id.* at 6:11-13. The specification goes

on to provide specific examples of CSEs that represent extensions or optimizations to file system or network functionalities, such as services to "[a]ccess files that reside in different locations" and network optimizations including "[m]odified protocol processing for custom hardware services." *Id.* at 6:14-28. In each case, software designed to rely on these services plainly would not function in its intended manner without them.

All of this intrinsic evidence guides a POSITA's understanding of what services are "critical" and confirms the definiteness of the claim scope. Amazon points to no evidence that any *other* understanding of "critical" would even be considered by a POSITA in the context of the '058 Patent and the above-cited intrinsic evidence. At the very least, Amazon's failure even to mention this evidence confirms that Amazon cannot prove indefiniteness by clear and convincing evidence, as required.

B. "functional replicas" (claim 1)

Plaintiff's Proposed Construction	Defendants' Proposed Construction
No construction necessary; plain and ordinary meaning.	Indefinite

Amazon argues that "replica" has a lexicographic definition, *i.e.* "a CSE having similar attributes to, but not necessarily and preferably not an exact copy of a CSE in the operating system (OS)," and that definition is indefinite as a term of degree because of the word "similar." Dkt. 71 at 10. This argument fails at both steps.

The Federal Circuit has explained that "[b]ecause language is limited, we have rejected the proposition that claims involving terms of degree are inherently indefinite." *Sonix Tech. Co. v. Publications Int'l, Ltd.*, 844 F.3d 1370, 1377 (Fed. Cir. 2017). "Thus, a patentee need not define his invention with mathematical precision in order to comply with the definiteness requirement." *Id.* (internal quotation marks omitted). "Claim language employing terms of degree has long been

found definite where it provided enough certainty to one of skill in the art when read in the context of the invention." *Interval Licensing LLC v. AOL, Inc.*, 766 F.3d 1364, 1370 (Fed. Cir. 2014). In determining whether the patent has provided sufficient guidance for a term of degree, a reviewing court should "look to the written description for guidance." *Id.* at 1371.

First, Amazon's focus on a single sentence from the patent specification ignores the claim context and the full disclosure of the patent specification. In particular, the claim term is "functional replica," not "replica." Even if the generic description of "replica" were indefinite (it is not), the limitation to functional replicas provides important clarification.

The specification contains an additional description of the scope of "the term replica" specifically in the context of *functional* replicas: "The CSE library includes replicas or substantial functional equivalents or replacements of kernel functions. The term replica, shall encompass any of these meanings, and although not a preferred embodiment, may even be a copy of a CSE that is part of the OS." '058 Patent at 8:27-32; *see also id.* at 9:52-56 ("The term replication means that like services are supplied [*i.e.*, that] essentially a same functionality is provided."). These sentences explicitly state what scope is "encompass[ed]" by "the term replica": (1) substantial functional equivalents of kernel functions; (2) replacements of kernel functions; and (3) copies of OSCSEs (*i.e.*, kernel functions). Of these three categories, "substantial functional equivalents" is logically the broadest, since either a replacement or a copy of a kernel function/OSCSE would necessarily also be functionally equivalent.

Accordingly, the phrase "functional replica" does not require mere similarity, but rather (at a minimum) "substantial functional equivalen[ce]." '058 Patent at 8:27-32. Amazon does not and cannot contend that determining the substantial functional equivalence of two CSEs is indefinite.

Indeed, juries are regularly required to determine functional equivalence in the context of the Doctrine of Equivalents or in the context of 35 U.S.C. § 112, ¶ 6.

C. "shared library" (claim 1)

Plaintiff's Proposed Construction	Defendants' Proposed Construction
No construction necessary; plain and ordinary meaning. In the alternative: An application library ***whose*** code space ***is*** shared among all user mode applications.	An application library code space shared among all user mode applications.

The term "shared library" appears throughout the specification and claims of the '058 Patent. It has a plain and ordinary meaning that is confirmed by the claim context and by the specification. Instead of recognizing this plain and ordinary meaning, Amazon demands including a confusing, circular "definition" that introduces additional terms unfamiliar to a jury and that, appears to contradict the claim context.

For example, the patent specification makes clear that "code space" refers to where a library is located, not to the library itself. *See, e.g.*, '058 Patent at 3:39-45 ("the same set of instructions in the same physical memory space, ***that is, shared code space***…"); *id.* at 6:54-55 ("Static library: An application ***whose*** code space is ***contained*** in a single application"); *id*. at 7:3-5 ("[W]hat is commonly done is to provide an application library ***in*** shared code space, which multiple applications can access."). This usage, which reflects the plain and ordinary meaning of "code space" to a POSITA, contradicts the notion that a shared library is *defined* as "an application library code space" as Amazon requests.

There is a simple explanation for the confusing construction, though: the patent applicant obviously introduced a pair of typographical errors into the definition of "Shared library." The original version of this definition, in the provisional application to which the '058 Patent claims priority, is shorter: "An application library ***whose*** code space ***is*** shared among all user mode

applications." Ex. 1 (Provisional Patent Application No. 60/504,213) at 9. That definition cleanly flowed from the definition of "Application library" above it, and paralleled the definition of "Static library" below it, confirming that the key difference between a shared library and a static library is whether the code space is contained in a single application or shared among applications:

> Application library: A collection of functions in an archive format that is combined with an application to export system elements.
>
> Shared library: An application library whose code space is shared among all user mode applications.
>
> Static library: An application library whose code space is contained in a single application.

Id.

When the applicant revised the provisional specification to form the non-provisional application, additional detail was added to the definition, but the words "whose" and "is" were removed. Those words were not deleted from the definition of "Static library," which retains the same definition in the final specification. The new language includes "The code space is different than that occupied by the kernel," confirming that "code space" is a space occupied by code, not code itself. This confirms that the deletion of "whose" was unintentional, and that the correct interpretation should retain the original language of the provisional. A POSITA reading the specification would readily understand that this is the correct interpretation. Therefore, if the extent the Court believes construction is necessary, the correct definition without the typographical errors should be included: "An application library ***whose*** code space ***is*** shared among all user mode applications."

D. "forms a part of … software applications" (claim 1)

Plaintiff's Proposed Construction	Defendants' Proposed Construction
No construction necessary; plain and ordinary meaning.	resides in the same address space as application code

Amazon's only asserted basis for this construction is prosecution disclaimer. Dkt. 71 at 12-13. The applicant distinguished prior art on the basis that the prior art's disclosure of a "proxy" was not an SLCSE because it did not "reside in the same address space as application code." In context, this simply states that a logical prerequisite for the "forms a part" limitation is not met; something cannot form a part of the software application if it does not even share an address space with that software application. The applicant's statement is not definitional, *i.e.* does not state or logically imply that "resides in the same address space as application code" is coextensive with the "forms a part" limitation. Amazon's proposal risks reading "forms a part" out of the claim entirely, and should be rejected.

Dated: November 12, 2024

Respectfully submitted,

/s/ Reza Mirzaie
Reza Mirzaie
CA State Bar No. 246953
Marc A. Fenster
CA State Bar No. 181067
Neil A. Rubin
CA State Bar No. 250761
Amy E. Hayden
CA State Bar No. 287026
Jacob R. Buczko
CA State Bar No. 269408
James S. Tsuei
CA State Bar No. 285530
James A. Milkey
CA State Bar No. 281283
Christian W. Conkle
CA State Bar No. 306374
Jonathan Ma

CA State Bar No. 312773
Daniel Kolko
CA State Bar No. 341680
RUSS AUGUST & KABAT
12424 Wilshire Boulevard, 12th Floor
Los Angeles, CA 90025
Telephone: 310-826-7474
Email: rmirzaie@raklaw.com
Email: mfenster@raklaw.com
Email: nrubin@raklaw.com
Email: ahayden@raklaw.com
Email: jbuczko@raklaw.com
Email: jtsuei@raklaw.com
Email: jmilkey@raklaw.com
Email: cconkle@raklaw.com
Email: jma@raklaw.com
Email: dkolko@raklaw.com

Qi (Peter) Tong
4925 Greenville Ave., Suite 200
Dallas, TX 75206
Email: ptong@raklaw.com

ATTORNEYS FOR PLAINTIFF VIRTAMOVE, CORP.

CERTIFICATE OF SERVICE

I certify that this document is being served upon counsel of record for Defendants on November 12, 2024 via electronic service.

/s/ *Christian W. Conkle*

Exhibit 1

Please type a plus sign (+) inside this box → [+]

PTO/SB/16 (8-00)
Approved for use through10/31/2002. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

PROVISIONAL APPLICATION FOR PATENT COVER SHEET

This is a request for filing a PROVISIONAL APPLICATION FOR PATENT und r 37 CFR 1.53(c).

INVENTOR(S)		
Given Name (first and middle [if any])	Family Name or Surname	Residence (City and either State or Foreign Country)
Donn	Rochette	Fenton, Iowa, USA
Dean	Huffman	Kanata, Ontario, Canada
Paul	O'Leary	Kanata, Ontario, Canada

☐ *Additional inventors are being named on th____ separately numbered sheets attached hereto*

TITLE OF THE INVENTION (280 characters max)

USER MODE CRITICAL SYSTEM ELEMENTS AS SHARED LIBRARIES

Direct all correspondence to: **CORRESPONDENCE ADDRESS**

☒ Customer Number **000293** → *Place Customer Number Bar Code Label here*

OR *Type Customer Number here*

☐ Firm or Individual Name

Address

Address

City | State | ZIP

Country | Telephone | Fax

ENCLOSED APPLICATION PARTS *(check all that apply)*

☒ Specification *Number of Pages* **23** ☐ CD(s), Number

☒ Drawing(s) *Number of Sheets* **6** ☐ Other (specify)

☐ Application Data Sheet. See 37 CFR 1.76

METHOD OF PAYMENT OF FILING FEES FOR THIS PROVISIONAL APPLICATION FOR PATENT *(check one)*

☒ Applicant claims small entity status. See 37 CFR 1.27.

☒ A check or money order is enclosed to cover the filing fees

☒ The Commissioner is hereby authorized to charge filing fees or credit any overpayment to Deposit Account Number **04-1577**

☐ Payment by credit card. Form PTO-2038 is attached.

FILING FEE AMOUNT ($) **$80.00**

The PTO did not receive the following listed items(s) Small entity status

The invention was made by an agency of the United States Government or under a contract with an agency of the United States Government.

☒ No.

☐ Yes, the name of the U.S. Government agency and the Government contract number are: _____

Respectfully submitted,

SIGNATURE

TYPED or PRINTED NAME **Ralph A. Dowell**

TELEPHONE **(703) 415-2555**

Date **09/17/03**

REGISTRATION NO. *(if appropriate)* **26,868**

Docket Number: **14451PRO**

USE ONLY FOR FILING A PROVISIONAL APPLICATION FOR PATENT

This collection of information is required by 37 CFR 1.51. The information is used by the public to file (and by the PTO to process) a provisional application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 8 hours to complete, including gathering, preparing, and submitting the complete provisional application to the PTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, Washington, D.C. 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Box Provisional Application, Assistant Commissioner for Patents, Washington, D.C.

P19SMALL/REV05

USER MODE CRITICAL SYSTEM ELEMENTS AS SHARED LIBRARIES

Field of the Invention

The invention relates to computer software, and more specifically to software that affects and extends services exported through application libraries.

Background of the Invention

Computer systems are designed in such a way that software application programs share common resources. It is traditionally the task of the operating system to provide mechanisms to safely and effectively control access to shared resources.

In some cases the centralized control of elements critical to software applications creates a limitation caused by conflicts for shared resources. Two software applications that require the same file, yet each require a different version of the file will conflict. In the same manner two applications that require independent access to specific network services will conflict. The common solution to these situations is to place software applications that may potentially conflict on separate compute platforms.

Current state of the art defines two architectural approaches to the migration of system elements from an operating system into an application context. In one architectural approach, a single server operating system places critical system elements in the same process. Despite the flexibility offered, the system elements continue to represent a centralized control point. In the other architectural approach, a multiple server operating system places critical system elements in separate processes. While offering even

greater options this architecture has suffered performance and operational differences.

Summary of the Invention

In accordance with a first broad aspect, the invention provides a computing architecture that has an operating system kernel having critical system elements and adapted to run in kernel mode; and a shared library adapted to store replicas of at least some of the critical system elements, for use by the software applications in user mode executing in the context of the application. The critical system elements are run in a context of a software application.

In some embodiments, the computing architecture has application libraries accessible by the software applications and augmented by the shared library.

In some embodiments, the critical system elements are left in the operating system kernel.

In some embodiments, the critical system elements use system calls to access services in the operating system kernel.

In some embodiments, the operating system kernel has a kernel module adapted to serve as an interface between a service in the context of an application program and a device driver.

In some embodiments, the critical system elements in the context of an application program use system calls to access services in the kernel module.

In some embodiments, the kernel module is adapted to provide a notification of an interruption to a service in the context of an application program.

In some embodiments, the operating system kernel is adapted to provide interrupt handling capabilities to user mode CSEs.

In some embodiments, the interrupt handling capabilities are initialized through a system call.

In some embodiments, the kernel module comprises a handler which is installed for a specific device interrupt.

In some embodiments, the handler is called when an interrupt request is generated by a hardware device.

In some embodiments, the handler notifies the service in the context of an application through the use of an up call mechanism.

In some embodiments, function overlays are used to intercept software application accesses to operating system services.

In some embodiments, the operating system kernel is enabled when the software application is loaded into memory.

In some embodiments, the critical system elements stored in the shared library are linked to the software applications as the software applications are loaded.

In some embodiments, in a native form the critical system elements rely on kernel services supplied by the

operating system kernel for device access, interrupt delivery, and virtual memory mapping.

In some embodiments, the kernel services are replicated in user mode and contained in the shared library with the critical system elements.

In some embodiments, the kernel services comprise memory allocation, synchronization and device access.

In some embodiments, the kernel services that are platform specific are not replicated.

In some embodiments, the kernel services which are platform specific are called by a critical system element running in user mode.

In some embodiments, a user process running under the computing architecture has a respective one of the software applications, the application libraries, the shared library and the critical system elements all of which are operating in user mode.

In some embodiments, the software applications are provided with respective versions of the critical system elements.

In some embodiments, the system elements which are application specific reside in user mode, while the system elements which are platform specific reside in the operating system kernel.

In some embodiments, a control code is placed in kernel mode.

In some embodiments, the kernel module is adapted to enable data exchange between the critical service elements in user mode and a device driver in kernel mode.

In some embodiments, the data exchange uses mapping of virtual memory such that data is transferred both from the critical service elements in user mode to the device driver in kernel mode and from the device driver in kernel mode to the critical service elements in user mode.

In some embodiments, the kernel module is adapted to export services for device interface.

In some embodiments, the export services comprise initialization to establish a channel between a critical system element of the critical system elements in user mode and a device.

In some embodiments, the export services comprise transfer of data from a critical system element of the critical system elements in user mode to a device managed by the operating system kernel.

In some embodiments, the export services include transfer of data from a device to a critical system element of the critical system elements in user mode.

According to a second broad aspect, the invention provides an operating system comprising the above computing architecture.

According to a third broad aspect, the invention provides a computing platform comprising the above operating system and computing hardware capable of running under the operating system.

According to a fourth broad aspect, the invention provides a shared library accessible to software applications in a user mode, the shared library being adapted to store system elements which are replicas of systems elements of an operating system kernel and which are critical to the software applications.

According to a fifth broad aspect, the invention provides an operating system kernel having systems elements and adapted to run in a kernel mode and to replicate, for storing in a shared library which is accessible by software applications in user mode, system elements of the system elements which are critical to the software applications.

According to a sixth broad aspect, the invention provides an article of manufacture comprising a computer usable medium having computer readable program code means embodied therein for a computing architecture. The computer readable code means in said article of manufacture has computer readable code means for running an operating system kernel having critical system elements in kernel mode; and computer readable code means for storing in a shared library replicas of at least some of the critical system elements, for use by software applications in user mode. The critical system elements are run in a context of a software application.

Accordingly, elements critical to a software application are migrated from centralized control in an operating system into the same context as the application.

Advantageously, the invention allows specific operating system services to efficiently operate in the same context as a software application.

Critical system elements normally embodied in an operating system are exported to software applications through shared libraries. The shared library services provided in an operating system are used to expose these additional system elements.

Brief Description of the Drawings

Preferred embodiments of the invention will now be described with reference to the attached drawings in which:

Figure 1 is an architectural view of the traditional monolithic operating system;

Figure 2 is an architectural view of a multi-server operating system in which some critical system elements are removed from the operating system kernel and are placed in multiple distinct processes or servers;

Figure 3 is an architectural view of an embodiment of the invention;

Figure 4 is a functional view showing how critical system elements exist in the same context as an application;

Figure 5 is a block diagram showing a kernel module provided by an embodiment of the invention; and

Figure 6 shows how interrupt handling occurs in an embodiment of the invention.

Detailed Description of the Preferred Embodiments

Embodiments of the invention enable the replication of critical system elements normally found in an operating

system kernel to run in the context of a software application. Critical system elements are replicated through the use of shared libraries. Replication implies that system elements are not replaced from an operating system, rather they become separate extensions accessed through shared libraries.

By way of introduction, a number of terms will now be defined.

Critical System Element (CSE): Any service or part of a service, normally supplied by an operating system, that is critical to the operation of a software application.

Compute platform: The combination of computer hardware and a single instance of an operating system.

User mode: The context in which applications execute.

Kernel mode: The context in which the kernel portion of an operating system executes. In conventional systems, there is a physical separation enforced by hardware between user mode and kernel mode. Application code cannot run in kernel mode.

Application Programming Interface (API): An API refers to the operating system and programming language specific functions used by applications. These are typically supplied in libraries which applications link with either when the application is created or when the application is loaded by the operating system. The interfaces are described by header files provided with an operating system distribution. In practice, system APIs are used by applications to access operating system services.

Application library: A collection of functions in an archive format that is combined with an application to export system elements.

Shared library: An application library whose code space is shared among all user mode applications.

Static library: An application library whose code space is contained in a single application.

Kernel module: A set of functions that reside and execute in kernel mode as extensions to the operating system kernel. It is common in most systems to include kernel modules which provide extensions to the existing operating system kernel.

Up call mechanism: A means by which a service in kernel mode calls a function in a user mode application context. It is common to provide an up call mechanism within an operating system kernel.

Figure 1 shows a conventional architecture where critical system elements execute in kernel mode. Critical system elements are contained in the operating system kernel. Applications access system elements through application libraries.

In order for an application of Figure 1 to make use of a critical system element in the kernel, the application makes a call to the application libraries. It is impractical to write applications which handle CPU specific/operating specific issues directly. As such, what is commonly done is to provide an application library in shared code space which multiple applications can access. This provides a platform independent interface where applications can access critical system elements. When the application makes a call to a

critical system element through the application library, a system call may be used to transition from user mode to kernel mode. The application stops running as the hardware enters kernel mode. The operating system kernel then provides the required functionality. It is noted that each oval in Figure 1 represents a different context. There are two application contexts in the illustrated example and the operating system context is not shown as an oval but also has its own context.

Figure 2 shows a system architecture where critical system elements execute in user mode but in a distinct context from applications. Some critical system elements are removed from the operating system kernel. They reside in multiple distinct processes or servers as discussed in United States Provisional Patent Application entitled "Drag & Drop Application Management" which is incorporated herein by reference. The servers that export critical system elements execute in a context distinct from the operating system kernel and applications. These servers operate at a peer level with respect to other applications. Applications access system elements through application libraries. The libraries in this case communicate with multiple servers in order to access critical system elements. Thus in the illustrated example, there are two application contexts and two critical system element contexts. When an application needs to make use of a critical system element which is being run in user mode, a rather convoluted sequence of events must take place. Typically the application first makes a platform independent call to the application library. The application library in turn makes a call to the operating system kernel. The operating system kernel then schedules the server with the critical system element in a different user mode context. After the server completes the operation, a switch back to kernel mode is made which then responds back to the application

through the application library. Due to this architecture, such implementations may result in poor performance. Ideally, an application which runs on the system of Figure 1 should be able to run on the system of Figure 2 as well. However, in practice it is difficult to maintain the same characteristics and performance using such an architecture.

The invention is contrasted with both of these architectures in that critical system elements are not isolated in the operating system kernel in the case of a monolithic architecture (Figure 1), also they are not removed from the context of an application as is the case with a multi-server architecture (Figure 2). Rather they are replicated and embodied in the context of an application.

Figure 3 shows an architectural view of the overall operation of the invention. Multiple user processes execute above a single instance of an operating system. Software applications utilize shared libraries as is done in United States Provisional Patent Application entitled "SOFTWARE SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS" which is incorporated herein by reference. The standard libraries are augmented by an extension which contains critical system elements. Extended services are similar to those that appear in the context of the operating system kernel.

Figure 4 shows functionality above the operating system kernel all of which is run in user mode while the operating system kernel itself runs in kernel mode. The user mode functionality includes the user applications, the standard application libraries, and a new extended shared library provided by an embodiment of the invention. The extended shared library includes replicas of kernel functions. These functions can be directly called by the applications and as

such can be run in the same context as the applications. In preferred embodiments, the kernel functions which are included in the extended shared library are also included in the operating system kernel. Furthermore, there might be different versions of a given critical system element forming part of the extended shared library with different applications accessing these different versions within their respective context.

In preferred embodiments, the platform specific aspects of the critical system element are left in the operating system kernel, for example certain system calls. Then the critical system elements which are included in the extended shared library may still make use of the operating system kernel to implement these platform specific functions.

Figure 5 represents the function of the kernel module (described in more detail below). A critical system element in the context of an application program uses system calls to access services in the kernel module. The kernel module serves as an interface between a service in the application context and a device driver. Specific device interrupts are vectored to the kernel module. A service in the context of an application is notified of an interrupt by the kernel module.

Figure 6 represents interrupt handling. Interrupt handling is initialized through a system call. A handler contained in the kernel module is installed for a specific device interrupt. When an interrupt request is generated by a hardware device the handler contained in the kernel module will be called. The handler notifies a service in the context of an application through the use of an up call mechanism.

Function Overlays

A function overlay occurs when the implementation of a function that would normally be called is replaced such that an extension or replacement function is called instead. The invention uses function overlays to intercept software application accesses to operating system services. The overlay is accomplished by use of an ability supplied by the operating system to allow a library to be preloaded before other libraries are loaded. Such an ability is used to cause the loading process (performed by the operating system) to link the application to a library extension supplied by the invention rather than to the library that would otherwise be used.

The functions overlaid by the invention serve as extensions to operating system services. When a function is overlaid in this manner it enables the service defined by the API to be exported in an alternate manner than that provided by the operating system in kernel mode.

Critical System Elements

According to the invention, some system elements that are critical to the operation of a software application are replicated from kernel mode, by the operating system, into user mode in the same context as that of the application.

These system elements are contained in a shared library. As such they are linked to a software application as the application is loaded. Figure 3 shows that an extension library is utilized.

In its native form, as it exists in the operating system kernel, a CSE uses services supplied by the operating system kernel. In order for the CSE to be migrated to user

mode and operate effectively, the services that the CSE uses from the operating system kernel are replicated in user mode and contained in the shared library with the CSE itself. Services of the type referred to here include, but are not limited to, memory allocation, synchronization and device access. Preferably, as discussed above, platform specific services are not replicated, but rather are left in the operating system kernel. These will then be called by the critical system element running in user mode.

Figure 4 shows that the invention allows for critical system elements to exist in the same context as an application. These services exported by library extensions do not replace those provided in an operating system kernel. Thus, in Figure 4 the user process is shown to include the application itself, the regular application library, the extended library and the critical system element all of which are operating in user mode. The operating system kernel is also shown to include critical system elements. In preferred embodiments, the critical system elements which are included in user mode are replicas of elements which are still included in the operating system kernel.

As discussed previously, different applications may be provided with their own versions of the critical system elements. Advantageously, this can make it appear that multiple applications running on a given platform have their own operating system. However in reality, there is only one operating system with application specific components in user mode, and with non-application specific components only residing in the kernel mode. It is noted that this allows different applications running on a given platform to operate in secure separation without clashing for resources and without versioning problems of libraries. Previous attempts to address

this problem have included building hardware that is capable of running multiple versions of an operating system, and building a virtual machine in software which effectively allows each service to have its own operating system, but in software. The new solution provided by this embodiment of the invention yields the same benefits of these two other solutions, but without the requirement from multiple operating systems.

Kernel module

In some embodiments, control code is placed in kernel mode as shown in Figure 4. Figure 5 shows that a kernel module is used to augment device access and interrupt notification.

As a device interface the kernel module enables data exchange between a user mode CSE and a device driver in kernel mode. The exchange uses mapping of virtual memory such that data is transferred in both directions without a copy.

Services exported for device interface typically include:

- Initialization. Establish a channel between a CSE in user mode and a specific device. Informs the interrupt service that this CSE requires notification.

- Write data. Transfer data from a CSE to a device. User mode virtual addresses are converted to kernel mode virtual addresses.

- Read data. Transfer data from a device to a CSE. Kernel mode data is mapped into virtual addresses in user mode.

During initialization, interrupt services are informed that for specific interrupts, they should call a

handler in the kernel module. The kernel module handles the interrupt by making an up call to the critical system element.

Interrupts related to a device being serviced by a CSE in user mode are extended such that notification is given to the CSE in use.

As shown in Figure 6 a handler is installed in the path of an interrupt. The handler uses an up call mechanism to inform the affected services in user mode.

A user mode service enables interrupt notification through the use of an initialization function.

The general system configuration of the present invention discloses one possible implementation of the invention.

In some embodiments, the 'C' programming language is used but other languages can alternatively be employed.

Function overlays have been implemented through application library pre-load. A library supplied with the invention is loaded before the standard libraries, using standard services supplied by the operating system. This allows specific functions (APIs) used by an application to be overlaid or intercepted by services supplied by the invention.

Access from a user mode CSE to the kernel module, for device I/O and registration of interrupt notification, is implemented by allowing the application to access the kernel module through standard device interfaces defined by the operating system. The kernel module is installed as a normal device driver. Once installed applications are able to open a

device that corresponds to the module allowing effective communication as with any other device or file operation.

Numerous modifications and variations of the present invention are possible in light of the above teachings. It is therefore to be understood that within the scope of the appended claims, the invention may be practiced otherwise than as specifically described herein.

WE CLAIM:

1. A computing architecture comprising:

an operating system kernel having critical system elements and adapted to run in kernel mode; and

a shared library adapted to store replicas of at least some of the critical system elements, for use by software applications in user mode;

wherein the critical system elements are run in a context of a software application.

2. A computing architecture according to claim 1 comprising application libraries accessible by the software applications and augmented by the shared library.

3. A computing architecture according to claim 1 wherein the critical system elements are left in the operating system kernel.

4. A computing architecture according to claim 1 wherein the critical system elements use system calls to access services in the operating system kernel.

5. A computing architecture according to claim 1 wherein the operating system kernel comprises a kernel module adapted to serve as an interface between a service in the context of an application program and a device driver.

6. A computing architecture according to claim 5 wherein the critical system elements in the context of an application program use system calls to access services in the kernel module.

7. A computing architecture according to claim 5 or 6 wherein the kernel module is adapted to provide a notification of an interruption to a service in the context of an application program.

8. A computing architecture according to any one of claims 5 to 7 wherein the operating system kernel is adapted to provide interrupt handling capabilities to user mode CSEs.

9. A computing architecture according to claim 8 wherein the interrupt handling capabilities are initialized through a system call.

10. A computing architecture according to claim 9 wherein the kernel module comprises a handler which is installed for a specific device interrupt.

11. A computing architecture according to claim 10 wherein the handler is called when an interrupt request is generated by a hardware device.

12. A computing architecture according to claim 11 wherein the handler notifies the service in the context of an application through the use of an up call mechanism.

13. A computing architecture according to any one of claims 1 to 12 wherein function overlays are used to intercept software application accesses to operating system services.

14. A computing architecture according to any one of claims 1 to 13 wherein the critical system elements stored in the shared library are linked to the software applications as the software applications are loaded.

15. A computing architecture according to any one of claims 1 to 14 wherein in a native form the critical system elements rely on kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.

16. A computing architecture according to claim 15 wherein the kernel services are replicated in user mode and contained in the shared library with the critical system elements.

17. A computing architecture according to claim 15 or 16 wherein the kernel services comprise memory allocation, synchronization and device access.

18. A computing architecture according to any one of claims 15 to 17 wherein the kernel services that are platform specific are not replicated.

19. A computing architecture according to claim 18 wherein the kernel services which are platform specific are called by a critical system element running in user mode.

20. A computing architecture according to claim 2 wherein a user process running under the computing architecture comprises a respective one of the software applications, the application libraries, the shared library and the critical system elements all of which are operating in user mode.

21. A computing architecture according to any one of claims 1 to 20 wherein the software applications are provided with respective versions of the critical system elements.

22. A computing architecture according to claim 21 wherein the system elements which are application specific

reside in user mode, while the system elements which are platform specific reside in the operating system kernel.

23. A computing architecture according to any one of claims 1 to 22 wherein a control code is placed in kernel mode.

24. A computing architecture according to claim 5 wherein the kernel module is adapted to enable data exchange between the critical service elements in user mode and a device driver in kernel mode.

25. A computing architecture according to claim 24 wherein the data exchange uses mapping of virtual memory such that data is transferred both from the critical service elements in user mode to the device driver in kernel mode and from the device driver in kernel mode to the critical service elements in user mode.

26. A computing architecture according to claim 24 or 25 wherein the kernel module is adapted to export services for device interface.

27. A computing architecture according to claim 26 wherein the export services comprise initialization to establish a channel between a critical system element of the critical system elements in user mode and a device.

28. A computing architecture according to claim 26 wherein the export services comprise transfer of data from a critical system element of the critical system elements in user mode to a device managed by the operating system kernel.

29. A computing architecture according to claim 26 wherein the export services comprise transfer of data from a

device to a critical system element of the critical system elements in user mode.

30. A operating system comprising the computing architecture of any one of claims 1 to 29.

31. A computing platform comprising the operating system of claim 30 and computing hardware capable of running under the operating system.

32. A computing architecture according to any one of Figure 1 to 3.

33. A shared library accessible to software applications in a user mode, the shared library being adapted to store system elements which are replicas of systems elements of an operating system kernel and which are critical to the software applications.

34. An operating system kernel having systems elements and adapted to run in a kernel mode and to replicate, for storing in a shared library which is accessible by software applications in user mode, system elements of the system elements which are critical to the software applications.

35. A computing architecture according to claim 13 wherein the operating system kernel is enabled when the software application is loaded into memory.

36. An article of manufacture comprising:

a computer usable medium having computer readable program code means embodied therein for a computing architecture, the computer readable code means in said article of manufacture comprising:

computer readable code means for running an operating system kernel having critical system elements in kernel mode; and

computer readable code means for storing in a shared library replicas of at least some of the critical system elements, for use by software applications in user mode;

wherein the critical system elements are run in a context of a software application.

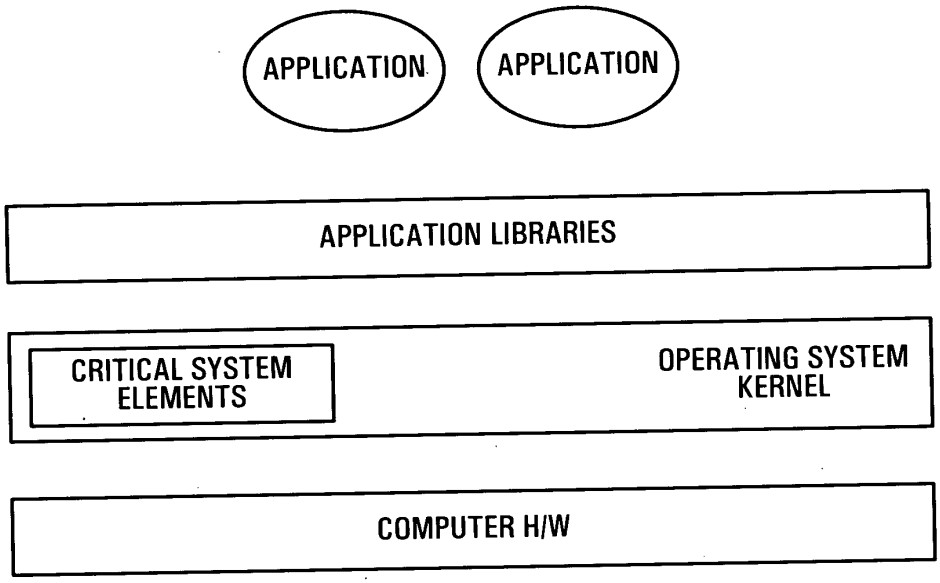
APPLICATION
APPLICATION
APPLICATION LIBRARIES
CRITICAL SYSTEM ELEMENTS
OPERATING SYSTEM KERNEL
COMPUTER H/W

FIG. 1

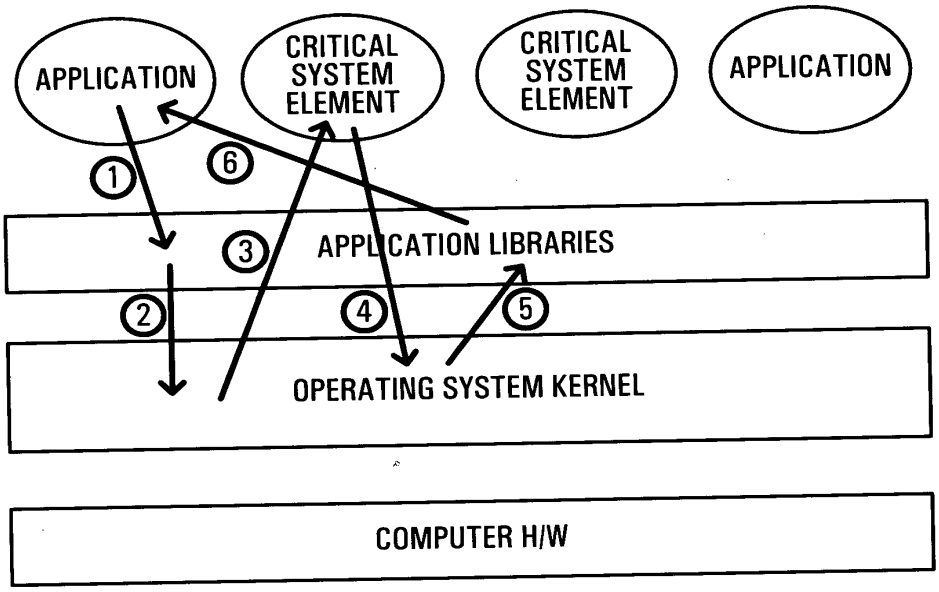
APPLICATION
CRITICAL SYSTEM ELEMENT
CRITICAL SYSTEM ELEMENT
APPLICATION
1
6
3
APPLICATION LIBRARIES
2
4
5
OPERATING SYSTEM KERNEL
COMPUTER H/W

FIG. 2

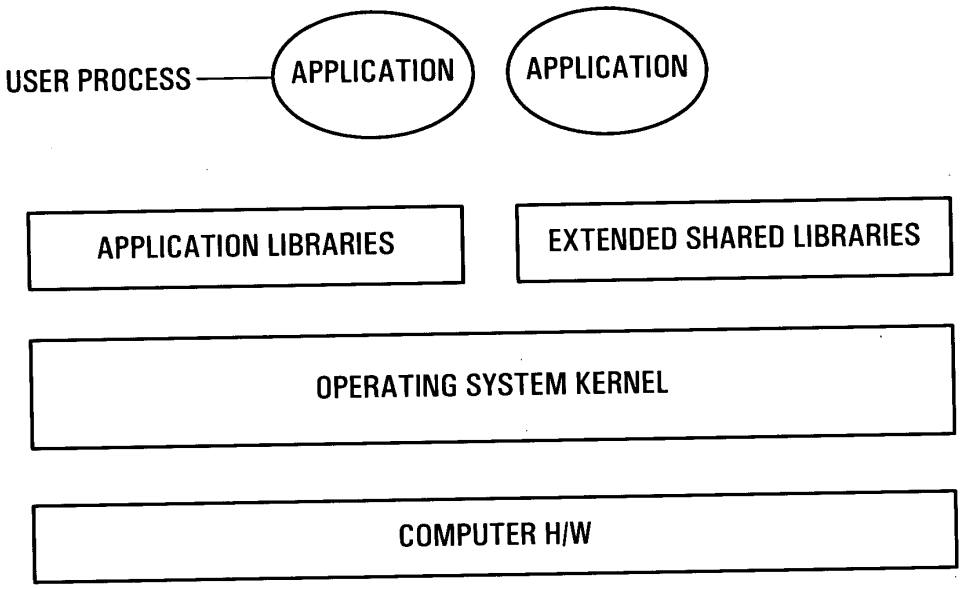
USER PROCESS
APPLICATION
APPLICATION
APPLICATION LIBRARIES
EXTENDED SHARED LIBRARIES
OPERATING SYSTEM KERNEL
COMPUTER H/W

FIG. 3

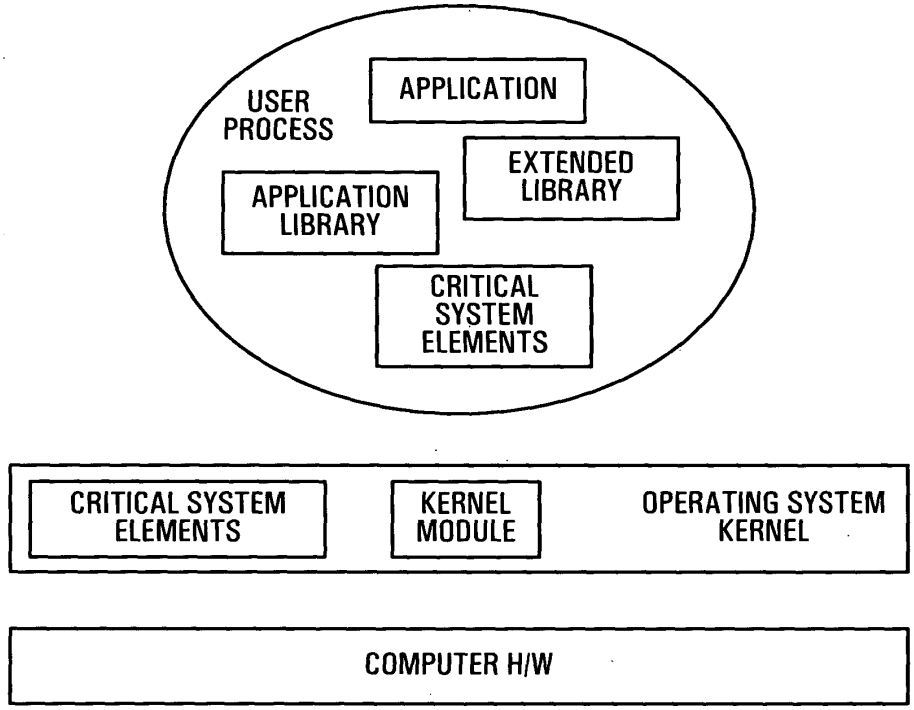
USER PROCESS
APPLICATION
EXTENDED LIBRARY
APPLICATION LIBRARY
CRITICAL SYSTEM ELEMENTS
CRITICAL SYSTEM ELEMENTS
KERNEL MODULE
OPERATING SYSTEM KERNEL
COMPUTER H/W

FIG. 4

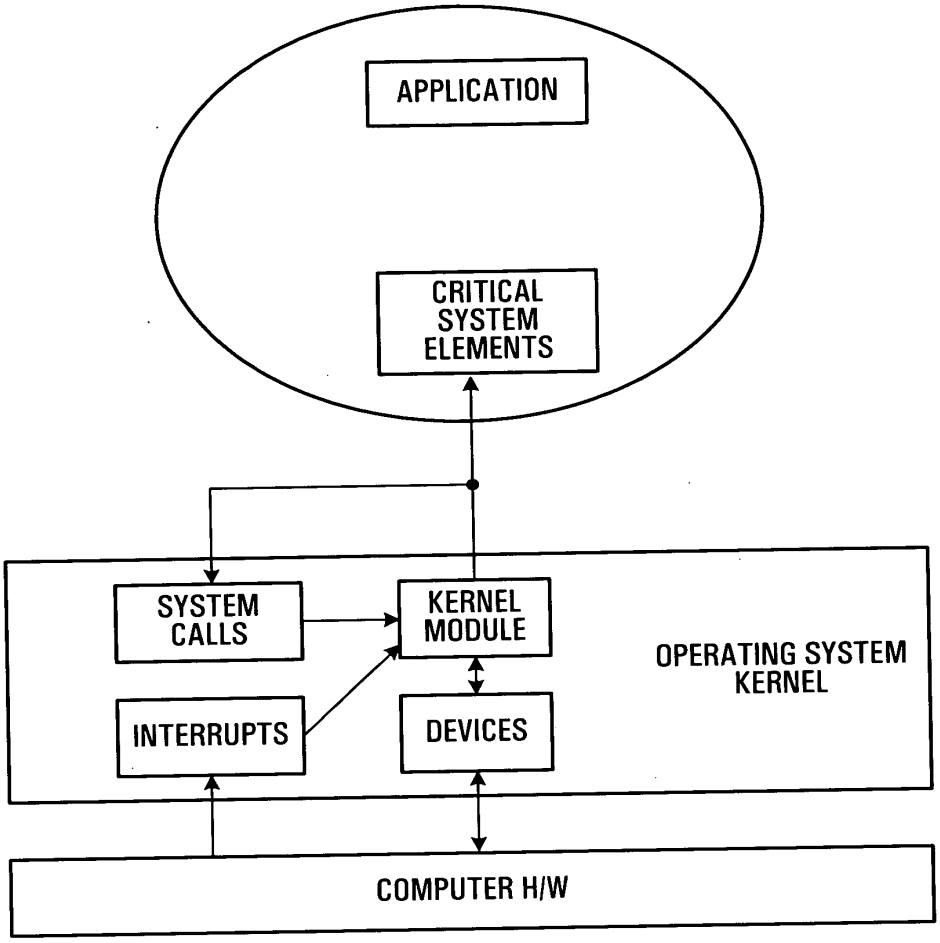
APPLICATION
CRITICAL SYSTEM ELEMENTS
SYSTEM CALLS
KERNEL MODULE
OPERATING SYSTEM KERNEL
INTERRUPTS
DEVICES
COMPUTER H/W

FIG. 5

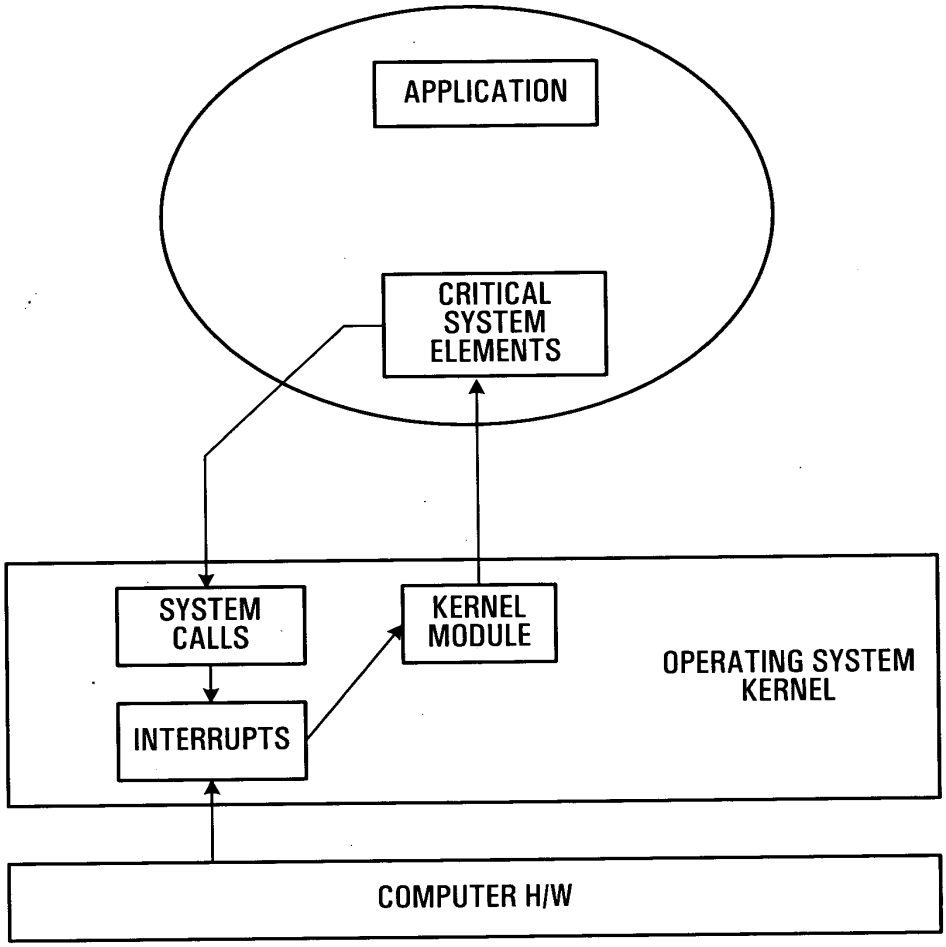
APPLICATION
CRITICAL SYSTEM ELEMENTS
SYSTEM CALLS
KERNEL MODULE
INTERRUPTS
OPERATING SYSTEM KERNEL
COMPUTER H/W

FIG. 6

PATENT APPLICATION SERIAL NO. _____

U.S. DEPARTMENT OF COMMERCE
PATENT AND TRADEMARK OFFICE
FEE RECORD SHEET

09/25/2003 EFLORES 00000027 60504213
01 FC:2005 80.00 OP

PTO-1556
(5/87)

*U.S. Government Printing Office: 2002 — 489-267/69033